IMPLEMENTASI PERANGKAT LUNAK SVMRK PADA SISTEM PARALEL MPI-LINUX

Alhadi Bustamam[1], Heru Suhartanto[2], T. Basaruddin[3]

1. Jurusan Matematika FMIPA-Universitas Indonesia
E-mail : alhadi@makara.cso.ui.ac.id
2. Fakultas Ilmu Komputer Universitas Indonesia
E-mail : heru@cs.ui.ac.id
3. Fakultas Ilmu Komputer Universitas Indonesia
E-mail : chan@cs.ui.ac.id

Abstrak

Pada makalah ini akan dibahas perangkat lunak SVMRK yang merupakan solver yang dikembangkan untuk menyelesaikan sistem persamaan linier yang kaku berdasarkan metode iteratif paralel implisit MRK (iPIMRK) dengan menggunakan implementasi ukuran langkah berubah-ubah (variable stepsize), seperti yang telah bahas oleh Bustamam dan Suhartanto et. al [1,2]. Proses integrasi metode iPIMRK ini menggunakan dua macam teknik perhitungan parameter metode yaitu: ***koefisien dari parameter metode bersifat konstan*** *(constant coefficients-FC) pada setiap langkah dan* ***koefisien dari parameter metode bersifat berubah-ubah*** *(variable coefficients-VC) pada setiap langkah. Implementasi perangkat lunak SVMRK ini merupakan modifikasi dan implementasi ulang terhadap perangkat lunak SMRK dari Suhartanto [2], dimana implementasi SMRK menggunakan FORTRAN 90 (F90) dan dijalankan di mesin paralel SGI_ORIGIN 2000, sedangkan modifikasi ulang menggunakan MPI-FORTRAN 77 (MPIF77) dengan penambahan modul-modul program paralelisasi agar bisa dijalankan di sistem paralel MPI-Linux yang ada di Lab HPCCSUI Fakultas Ilmu Komputer UI. Pada makalah ini akan ditampilkan skema, implementasi paralelisasi dan implementasi modul-modul program, kompilasi dan teknik menjalankan aplikasi secara SPMD dan hasil eksperimen numerik yang diperoleh dan evaluasi kinerja metode iteratif paralel implisit multistep Runge-Kutta untuk menyelesaikan sistem persamaan diferensial biasa berupa persoalan dense yang bersifat kaku (stiff). Dari hasil percobaan diperoleh bahwa metode dengan koefisien parameter berubah memiliki keunggulan dari sisi speed-up, efisiensi dan akurasi, tetapi lebih lama waktu komputasinya dibandingkan metode dengan koefisien parameter konstan.*

Kata Kunci: mpi, spmd, sistem persamaan diferensial biasa, persoalan dense yang stiff, metode iteratif paralel implisit multistep runge-kutta, sistem paralel mpi-linux.

1. Pendahuluan

Perangkat lunak SVMRK merupakan *solver* yang dikembangkan untuk menyelesaikan sistem persamaan linier yang *stiff* berdasarkan metode iteratif paralel implisit MRK (iPIMRK) dengan menggunakan implementasi ukuran langkah berubah-ubah (*variable stepsize*) yang didefinisikan secara iteratif sebanyak L kali sebagai berikut ini, lihat Bustamam dan Suhartanto et. al [1,2]:

$$\begin{aligned} Y_n^{(j)} &= (A \otimes I_m) y^{(n)} + h_n([B-W] \otimes I_m) F(Y_n^{(j-1)}) \\ &\quad + h_n(W \otimes I_m) F(Y_n^{(j)}), j = 1,\ldots,L \\ y_{n+1} &= (e_s^T \otimes I_m) F(Y_n^{(L)}) \end{aligned} \quad (1)$$

dalam hal ini, $Y_n^{(0)}$ adalah nilai aproksimasi awalan untuk solusi intermediate $Y_n \in \Re^{sm}$, matrik W adalah matrik pemecah (spliting matrix) terhadap matrik B, yang terdiri atas nilai-nilai eigen nonnegatif sembarang dari matrik B, $Y_n \in \Re^m$ adalah solusi yang dicari.

Pemilihan nilai matrik W ini dilakukan sedemikian hingga agar proses iterasi untuk masing-masing $Y_{ni}, i = 1, \ldots, s$ dapat dilakukan secara paralel, misalnya dengan menggunakan W=D

(matrik diagonal) disebut metode iteratif paralel implisit diagonal MRK (iPDIMRK) atau menggunakan W=T (matrik triangular) disebut metode iteratif paralel implisit triangular (iPTIMRK). Bentuk matrik W yang digunakan pada pembahasan ini adalah matrik segitiga bawah T yang didapat dari dekomposisi Crout terhadap matrik B. Pada tahap implementasinya, proses integrasi metode iPIMRK ini menggunakan dua macam teknik perhitungan parameter metode yaitu: koefisien dari parameter $s\ i\ Y\ i\ n$,..., 1 , , = 1 metode bersifat konstan (constant coefficients-FC) pada setiap langkah dan koefisien dari parameter metode bersifat berubah-ubah (variable coefficients-VC) pada setiap langkah, lihat Burrage dan Suhartanto[3,4].
Berdasarkan persamaan (1) maka algoritma umum metode iPIMRK didefinisikan sebagai berikut:

Tabel 1 : Algoritma Umum Metode iPIMRK

Misalkan $Z = Y_n^j - (A \otimes I_m) y^{(n)}$
Iterasi lengkapnya dapat ditulis sbb :
For $j = 1, \ldots, L$ ← OUTER iteration
  Compute $G(Y_n^{(j-1)}) = h([B - W] \otimes I_m) F(Y_n^{(j-1)})$
  For $k = 1, \ldots, inner$ ← INNER iteration
    $(I_{sm} - W \otimes hJ_m)\Delta Z = -[Z^{(k-1)} - G(Y_n^{(j-1)}) - h(W \otimes I_m) F(Z^{(k-1)} + A \otimes I_m) y^n)]$
    $Z^k = Z^{(k-1)} + \Delta Z$
  end for
  $Y_n^{(j)} = Z^{(niner)} + (A \otimes I_m) y^{(n)}$
  $y_{n+1}^{(j)} = (e_8^T \otimes I_m) Y_n^{(j)} = O(h^{q+j})$
end for

Berdasarkan algoritma umum pada Tabel 1, pada metode iPIMRK ini dapat dilakukan tiga proses paralel yang secara umum dapat diuraikan sebagai berikut:

1. ***Parallel_Stages***: Pehitungan nilai evaluasi fungsi pada saat iterasi ke L untuk masing-masing *stages* secara paralel di sejumlah *stages* prosesor.
2. ***Parallel_Factors***: Faktorisasi matrik iterasi (ITERM) dari sistem persamaan linier pada setiap *stages* secara paralel di sejumlah *stages* prosesor dengan menggunakan input matrik Jacobian (JAC), H dan matrik W (DTI). Hasil komputasi dan vektor pivot PIV berguna untuk pencarian solusi dari sistem persamaan linier tersebut secara paralel pada masing-masing *stages* oleh modul *parallel_solves*.
3. ***Parallel_Solves***: Mencari vektor solusi . dari sistem persamaan linier yang terdiri atas matrik faktorisasi M dan matrik sisi kanan W0 dengan vektor pivot PIV secara paralel pada sejumlah *stages* prosesor untuk masing-masing stages.

Implementasi perangkat lunak SVMRK ini merupakan modifikasi dan implementasi ulang terhadap perangkat lunak SMRK dari Suhartanto dan Burrage [11,12]. Implementasi SMRK menggunakan FORTRAN 90 (F90) dan dijalankan di mesin paralel SGI_ORIGIN 2000, sedangkan modifikasi ulang menggunakan GNU-FORTRAN 77 (yang diadopsi oleh MPI[7,9] menjadi MPIF77) dengan penambahan modul-modul program paralelisasi agar bisa dijalankan di sistem paralel MPILinux yang ada di Lab HPCCSUI Fakultas Ilmu Komputer UI, lihat Bustamam dan Suhartanto et. al. [1,2].

Kegunaan dan fungsi-fungsi yang ada di modul-modul versi MPIF77 ini sama dengan kegunaan dan fungsi-fungsi yang ada di modul-modul versi F90 dari Suhartanto[11] dengan perbedaan implementasi dari sisi teknik paralelisasi, struktur data dan teknik pemrogramannya. Perbedaan tersebut terutama disebabkan oleh karena banyaknya keterbatasan dari bahasa GNU-FORTRAN77 dibandingkan bahasa FORTAN90, lihat Burley[5], Koffman[8], Nyhoff[10]. Implementasi kembali modul-modul FORTRAN 90 ke GNU-FORTRAN77 membutuhkan beberapa modul-modul tambahan, khususnya menyangkut dengan struktur data dan konfigurasi dari matrik sistem.

2. Disain Internal Program

Berdasarkan algoritma umum metode iPIMRK pada tabel.1, maka secara umum implementasi metode iteratif paralel implisit MRK menggunakan algoritma pada Tabel 2.(a), 2.(b) dan 2.(c) berikut ini:

Tabel 2.(a) : Tugas yang dilakukan di prosesor#0 (*server*)

MODUL *DRIVER* (DR_SVMRK,DR_CUSP,DR_SBRUSS,DR_SDENSE)	
	PROSESOR #0
[1]	Lakukan inisialisasi MPI untuk proses paralel, inisialisasi persoalan & data, dimensi *working-matrix*, parameter metode, predictor, serta variable-variable yang relevan.
[2]	Jika jumlah prosesor >= jumlah stages maka **USE_PARALLEL=.TRUE.**, selain itu **USE_PARALLEL=.FALSE.**
[3]	Jika USE_PARALLEL kirim informasi dan data yang dibutuhkan ke seluruh PROSESOR #1,..,#stages-1. Jika tidak maka lakukan komputasi iPIMRK secara sekuensial di PROSESOR #0. Selesai jika {USE_PARALLEL}
[4]	CALL INTEG_MRK {INTEGRASI UTAMA di MODUL SVMRK}.
[5]	SELESAI INTEGRASI UTAMA
[6]	CETAK REPORT
[7]	Jika USE_PARALLEL kirim informasi TASK_INFO='SVMRK_FINISHED')keseluruh PROSESOR #1,…,stages-1 yang berarti bahwa proses integrasi telah selesai
[8]	FINALIZED proses MPI

Tabel 2.(b) : Tugas yang dilakukan di prosesor#lain (*nonserver*)

MODUL *DRIVER* (DR_SVMRK,DR_CUSP,DR_SBRUSS,DR_SDENSE)
PROSESOR #LAIN
[1] Jika USE_PARALLEL, terima informasi TASK_INFO(proses paralel yang akan dilakukan) & data dari PROSESOR#0.
[2] Jika TASK_INFO = 'SVMRK_Starting' atau 'SVMRK_Continue' maka USE_PARALEL=.TRUE.
[3] Selama USE_PARALLEL
A. Terima COMP_INFO {informasi komputasi yang sedang berjalan}
B. Jika COMP_INFO='SVMRK_Finished': USE_PARALLEL=.FALSE. {Keluar, proses paralel}
C. Jika COMP_INFO='Compute_Stages': Lakukan PARALLEL_STAGES
D. Jika COMP_INFO='Compute_Factors': Lakukan PARALLEL_FACTORS
E. Jika COMP_INFO='Compute_Solves': Lakukan PARALLEL_SOLVES Selesai (Selama USE_PARALLEL}
{SELESAI MODUL DRIVER iPIMRK}

2. Disain Internal Program

Berdasarkan algoritma umum metode iPIMRK pada tabel.1, maka secara umum implementasi metode iteratif paralel implisit MRK menggunakan algoritma pada Tabel 2.(a), 2.(b) dan 2.(c) berikut ini:

Tabel 2.(a) : Tugas yang dilakukan di prosesor#0 (*server*)

MODUL *DRIVER* (DR_SVMRK,DR_CUSP,DR_SBRUSS,DR_SDENSE)
PROSESOR #0
[1] Lakukan inisialisasi MPI untuk proses paralel, inisialisasi persoalan & data, dimensi *working-matrix*, parameter metode, predictor, serta variable-variable yang relevan.
[2] Jika jumlah prosesor >= jumlah stages maka **USE_PARALLEL=.TRUE.**, selain itu **USE_PARALLEL=.FALSE.**
[3] Jika USE_PARALLEL kirim informasi dan data yang dibutuhkan ke seluruh PROSESOR #1,..,#stages-1. Jika tidak maka lakukan komputasi iPIMRK secara sekuensial di PROSESOR #0. Selesai jika {USE_PARALLEL}
[4] CALL INTEG_MRK {INTEGRASI UTAMA di MODUL SVMRK}.
[5] SELESAI INTEGRASI UTAMA
[6] CETAK REPORT
[7] Jika USE_PARALLEL kirim informasi TASK_INFO='SVMRK_FINISHED')keseluruh PROSESOR #1,…,stages-1 yang berarti bahwa proses integrasi telah selesai
[8] FINALIZED proses MPI

Tabel 2.(b) : Tugas yang dilakukan di prosesor#lain (*nonserver*)

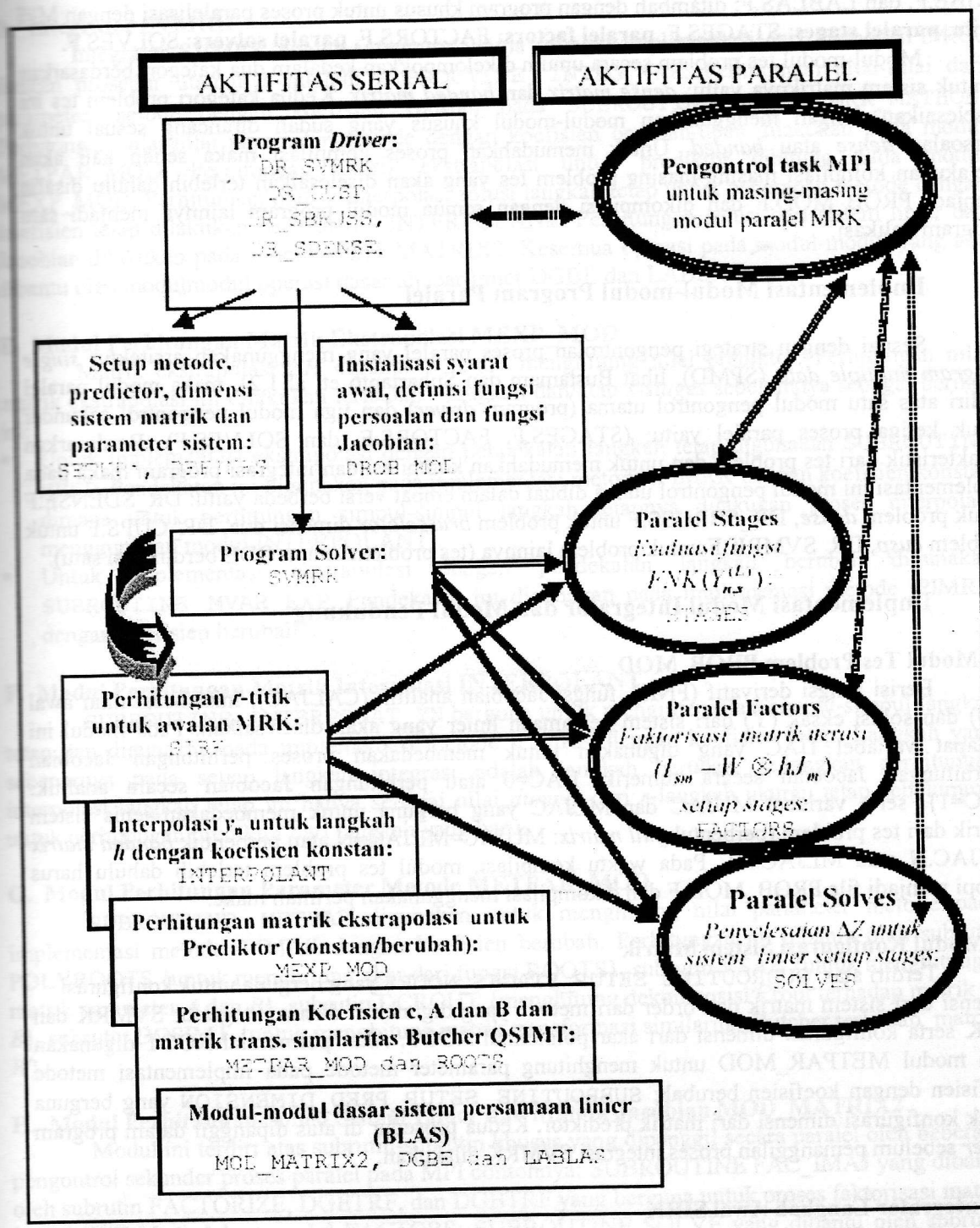
MODUL *DRIVER* (DR_SVMRK,DR_CUSP,DR_SBRUSS,DR_SDENSE)
PROSESOR #LAIN
[1] Jika USE_PARALLEL, terima informasi TASK_INFO(proses paralel yang akan dilakukan) & data dari PROSESOR#0.
[2] Jika TASK_INFO = 'SVMRK_Starting' atau 'SVMRK_Continue' maka USE_PARALEL=.TRUE.
[3] Selama USE_PARALLEL A. Terima COMP_INFO {informasi komputasi yang sedang berjalan} B. Jika COMP_INFO='SVMRK_Finished': USE_PARALLEL=.FALSE. {Keluar, proses paralel} C. Jika COMP_INFO='Compute_Stages': Lakukan PARALLEL_STAGES D. Jika COMP_INFO='Compute_Factors': Lakukan PARALLEL_FACTORS E. Jika COMP_INFO='Compute_Solves': Lakukan PARALLEL_SOLVES Selesai (Selama USE_PARALLEL}
{SELESAI MODUL DRIVER iPIMRK}

Tabel 2.(c) : Integrator SVMRK di prosesor#0 (*server*)

MODUL INTEGRATOR SVMRK
PROSESOR #0 [1] CALL INTEG_IRK untuk menghitung solusi dan langkah awal. [2] Gunakan matrik ekstrapolasi konstan jika matrik kstrapolasi berubah-ubah gagal dibentuk pada setiap langkah. [3] Mulai ITEGRASI UTAMA a. Jalankan METPAR dan set STIFF=.TRUE., Jika gagal gunakan parameter metode konstan. b. Jika dibutuhkan hitung JACOBIAN c. Jika dibutuhkan bentuk matrik iterasi dan lakukan FAKTORISASI: Jika dibutuhkan lakukan **PARALLEL_FACTORS** d. Jika dibutuhkan bentuk matrik ekstrapolasi • Set *Predictor* • Lakukan evaluasi fungsi: Jika dibutuhkan lakukan **PARALLEL_STAGES** • Mulai proses ITERASI_LUAR (*outer iteration*) o Lakukan Iterasi Newton (*inner iteration*) ✓ Jika dibutuhkan lakukan evaluasi fungsi: Jika dibutuhkan lakukan **PARALLEL_STAGES** ✓ Selesaikan sistem linier Newton: Jika dibutuhkan lakukan **PARALLEL_SOLVES** o Hitung THETA o Jika DIVERGEN maka KELUAR *outer iteration*. o Hitung estimasi kesalahan lokal o Jika VARIABLE ORDER atau kesalahan cukup kecil maka KELUAR *outer iteration* o Lakukan evaluasi fungsi: Jika dibutuhkan lakukan PARALEL_STAGES • Selesai proses ITERASI_LUAR • Jika NOT(DIVERGEN) maka o Hitung ukuran langkah berikutnya o Jika ukuran langkah diterima (*accepted*) maka lakukan beberapa modifikasi data untuk informasi yang dibutuhkan ✓ Set JACOBIAN *flag* Jika tidak, ✓ Set JACOBIAN *flag* Selesai jika {ukuran langkah diterima}. Jika tidak, o Perbarui ukuran langkah o Set JACOBIAN *flag* Selesai {jika NOT(DIVERGEN)} [4] Selesai {ITEGRASI UTAMA}

3. Skema Implementasi Modul-Modul Program

Secara umum skema modul-modul program MPIF77 dari implementasi perangkat lunak SVMRK pada lingkungan komputasi paralel MPI-Linux adalah sebagai berikut:



Gambar 1: Skema modul-modul program metode iPIMRK pada sistem paralel MPI-Linux

Implementasi dari skema modul-modul program pada gambar 1 dalam MPIF77 berbentuk file-file program GNU-FORTRAN77(*.F) yang terdiri atas: **program *driver***: DR_SDENSE.F (untuk tes problem *dense*), DR_SBRUSS.F (untuk tes problem *brusselator*), DR_CUPS.F (untuk test problem *cusp*), DR_SVMRK.F (untuk tes problem ODE 1 dimensi standar); **program tes**

problem: PROB_MOD.F, terdiri atas beberapa modul problem tes (misalnya: PROB_HIRES.F, PROB_ROB.F, PROB_B2.F, PROB_CUSP.F, PROB_DENSE.F, PROB_BRUSS.F, dan lain-lain); **program integrator**: SVMRK.F dan SIRK.F; **program pendukung**: DEFINES.F, INTERPOLANT.F, MEXP_MOD.F, ROOTS.F, METPAR_MOD.F, MOD_MATRIX2.F, DGBE.F, dan LABLAS.F; ditambah dengan program khusus untuk proses paralelisasi dengan MPI yaitu, **paralel stages**: STAGES.F, **paralel factors:** FACTORS.F, **paralel solvers**: SOLVES.F.

Modul-modul tes problem secara umum dikelompokkan kedalam dua kategori berdasarkan bentuk sistem matriknya yaitu: *dense matrix* dan *banded matrix*. Kedua kategori problem tes ini diselesaikan dengan menggunakan modul-modul khusus yang sudah dirancang sesuai untuk persoalan *dense* atau *banded*. Untuk memudahkan proses kompilasi maka setiap kali akan melakukan kompilasi masing-masing problem tes yang akan diselesaikan terlebih dahulu disalin menjadi PROB_MOD.F dan dikompilasi dengan semua modul program lainnya menjadi satu program aplikasi.

4. Implementasi Modul-modul Program Paralel

Sesuai dengan strategi pengontrolan proses paralel yang menggunakan arsitektur *single program multiple data* (SPMD), lihat Bustamam dan Suhartanto et. al[1,2], maka modul paralel terdiri atas satu modul pengontrol utama (program *driver*) dan tiga modul pengontrol sekunder untuk ketiga proses paralel yaitu: (STAGES.F, FACTORS.F, dan SOLVES.F). Berdasarkan karakteristik dari tes problem dan untuk memudahkan kompilasi dan integrasi program maka pada implementasi ini modul pengontrol utama dibuat dalam empat versi berbeda yaitu: DR_SDENSE.F untuk problem *dense*, DR_SBRUSS.F untuk problem *brusselator* dimensi dua, DR_CUPS.F untuk problem *cusp*, DR_SVMRK.F untuk problem lainnya (tes problem umum ODE berdimensi satu).

5. Implementasi Modul Integrator dan Modul Pendukung

A. Modul Tes Problem PROB_MOD

Berisi fungsi derivatif (FNK), fungsi Jacobian analitik (CALJAC), inisialisasi nilai awal (Y0) dan solusi eksak (Y) dari sistem persamaan linier yang akan diselesaikan. Pada modul ini terdapat variabel IJAC yang digunakan untuk membedakan proses perhitungan Jacobian {perhitungan Jacobian secara numerik: IJAC=0 atau perhitungan Jacobian secara analitik: IJAC=1}, serta variabel MUJAC dan MLJAC yang berguna untuk membedakan jenis sistem matrik dari tes problem (berbentuk *full matrix*: MUJAC=MLJAC=N atau berbentuk *banded matrix* MUJAC.N atau MLJAC.N}. Pada waktu kompilasi, modul tes problem terlebih dahulu harus dikopi menjadi file PROB_MOD.F dan dikompilasi menggunakan perintah make.

B. Modul Konfigurasi Sistem Matrik

Terdiri atas: `SUBROUTINE SETUP_STAGES_NODES` yang berguna untuk konfigurasi dimensi dari sistem matrik dan order dari metode yang digunakan dalam integrator SVMRK dan SIRK serta konfigurasi dimensi dari akar polinomial ROOT. Akar polinomial ROOT digunakan oleh modul METPAR_MOD untuk menghitung parameter metode pada implementasi metode koefisien dengan koefisien berubah; `SUBROUTINE SETUP_PRED_DIMENSION` yang berguna untuk konfigurasi dimensi dari matrik prediktor. Kedua prosedur di atas dipanggil dalam program *driver* sebelum pemanggilan proses integrasi SVMRK dilakukan.

C. Integrator Langkah Awal SIRK

Integrator langkah awal SIRK berupa `SUBROUTINE INTEG_IRK` yaitu metode satu langkah dengan beberapa tahapan implisit Runge-Kutta (IRK) tipe Radau untuk persoalan yang kaku dengan orde ($2s$-1). Integrator SIRK digunakan oleh integrator SVMRK untuk menghitung solusi pada r-langkah awal sebagai langkah awalan dari metode MRK, setelah itu proses integrasi dilakukan integrator SVMRK sampai selesai. Untuk menjaga akurasi dan konvergensi pada tahap awal tersebut maka metode IRK yang digunakan seharusnya memiliki order yang lebih tinggi atau sama dengan order metode MRK. yang memanggil, contoh: metode MRK33 menggunakan IRK14

berorder 7, metode MRK22 menggunakan IRK13 berorder 5. Dalam implementasinya algoritma yang digunakan oleh IRK pada dasarnya sama dengan metode satu langkah MRK dengan mengambil nilai *r=1*. Nilainilai parameter metode SIRK yang digunakan dipanggil dari: `SUBROUTINE SETUP_IRK_METHOD`.

D. Integrator SVMRK

Integrator SVMRK adalah integrator utama sebagai implementasi dari metode iPIMRK dengan prosedur utamanya adalah: `SUBROUTINE INTEG_MRK`. Perhitungan nilai-nilai dari parameter metode dengan koefisien konstan pada `SUBROUTINE SETUP_MRK_METHOD`. Perhitungan nilai-nilai parameter metode dengan koefisien berubah-ubah dilakukan pada modul METPAR_MOD. Perhitungan matrik ekstrapolasi untuk prediktor dihitung pada modul MEXP_MOD. Perhitungan matrik interpolasi solusi langkah tetap untuk penerapan metode dengan koefisien tetap dilakukan pada modul INTERPOLANT. Perhitungan solusi dari sistem linier dan Jacobian dilakukan pada modul MOD_MATRIX2. Kesemua operasi pada modul-modul yang ada dibantu oleh modulmodul operasi dasar aljabar linier DGBE dan LABLAS.

E. Modul Perhitungan Matrik Ekstrapolasi MEXP_MOD

Matrik ekstrapolasi *AH* digunakan untuk menghitung nilai prediktor. Perhitungan nilai matrik ekstrapolasi ini dilakukan terhadap *r*-langkah dan/atau *s-stages* sebelumnya sebagai berikut ini:

- Untuk implementasi ekstrapolasi dengan pendekatan langkah tetap digunakan `SUBROUTINE MEXP`. Pendekatan ini digunakan pada implementasi metode iPIMRK dengan koefisien konstan dimana untuk perhitungan simpul-simpul langkah tetapnya dilakukan proses interpolasi menggunakan modul INTERPOLANT).
- Untuk implementasi ekstrapolasi dengan pendekatan langkah berubah digunakan `SUBROUTINE MVAR_EXP`. Pendekatan ini digunakan pada implementasi metode iPIMRK dengan koefisien berubah.

F. Modul Perhitungan Matrik Interpolasi INTERPOLANT

`SUBROUTINE INTERPOLAT_YN` berguna untuk menginterpolasi simpul-simpul langkah tetap dan digunakan pada implementasi metode iPIMRK dengan koefisien tetap. Langkah yang sebenarnya pada setiap langkah integrasi adalah langkah berubah, sedangkan perhitungan interpolasi langkah tetap ini hanya sebagai nilai *dummy* dari *r*-langkah ukuran tetap sebelumnya untuk perhitungan prediktor dari langkah yang baru.

G. Modul Perhitungan Parameter Metode METPAR_MOD

`SUBROUTINE METPAR` digunakan untuk menghitung nilai parameter metode pada implementasi metode iPIMRK dengan koefisien berubah. Perhitungan ini dibantu oleh subrutin POLYROOTS {untuk menghitung akar dari fungsi ROOTS}, subrutin VSAB {untuk menghitung matrik parameter *A* dan *B*}, subrutin DCROUT {menghitung dekomposisi Crout terhadap matrik *B*}, subrutin DQSIMT {untuk menghitung matrik transformasi similaritas Butcher terhadap matrik *W*}.

H. Modul Perhitungan Solusi Sistem Linier dan Matrik Jacobian MOD_MATRIX2

Modul ini terdiri atas subrutin-subrutin khusus yang dipanggil secara paralel oleh beberapa pengontrol sekunder proses paralel pada MPI contohnya: SUBROUTINE FAC_IMAJ yang dibantu oleh subrutin FACTORIZE, DGETRF, dan DGBTRF yang berguna untuk proses faktorisasi matrik iterasi ITERM di dalam modul FACTORS; SUBROUTINE SOLVE yang dibantu oleh subrutin DGETRS dan DGBTRS untuk perhitungan solusi sistem linier di dalam modul SOLVES. Subrutin DGETRF dan DGETRS digunakan untuk sistem matrik berbentuk *full matrix* sedangkan Subrutin DGBTRF dan DGBTRS digunakan untuk sistem matrik berbentuk *banded matrix*. Selain itu terdapat juga subrutin COMPJAC yang berguna untuk perhitungan matrik Jacobian secara numerik apabila nilai IJAC=0. Subrutin COMPJAC ini dapat digunakan baik untuk sistem dengan *full matrix* maupun sistem dengan *banded matrix*.

I. Modul Pendukung Operasi-operasi Dasar Matrik dan Aljabar Linier DGBE dan LABLAS.

Berisi subrutin-subrutin pendukung operasi matrik dan operasi aljabar linier seperti perkalian matrik (DGEMM), faktorisasi LU untuk matrik penuh (DGETRF) dan faktorisasi LU untuk matrik *banded* (DGBTRF) pada modul MOD_MATRIX2, penyelesaian sistem persamaan linier matrik penuh (DGETRS) dan penyelesaian sistem persamaan linier matrik *banded* (DGBTRS) pada modul MOD_MATRIX2, perhitungan determinan atau invers matrik (DGEDI) menggunakan faktor dari hasil faktorisasi Gauss (DGEFA) pada modul METPAR_MOD dan MEXP_MOD, perhitungan nilai eigen (DGEEV) pada modul ROOTS, menghitung faktorisasi QR (DGEQRF) dan membuat matrik ortonormal Q (DORGQR) pada modul problem DENSE.

6. Kompilasi Program Aplikasi SVMRK

Kompilasi program aplikasi SVMRK dilakukan dengan perintah make berikut ini:

- {untuk program DR_SVMRK} : `make -f Makefile`
- {untuk program DR_CUSP} : `make -f Makecusp`
- {untuk program DR_SBRUSS} : `make -f Makebruss`
- {untuk program DR_SDENSE} : `make -f Makedense`

dimana file Makefile berisi konfigurasi untuk DR_SVMRK, file `Makebruss` berisi konfigurasi untuk DR_SBRUSS, file `Makecusp` berisi konfigurasi untuk DR_CUSP, dan file `Makedense` berisi konfigurasi untuk DR_SDENSE. Perbedaan isi dari keempat file konfigurasi hanya pada nama file *driver* yang digunakan.

7. Menjalankan Program Aplikasi SVMRK

Untuk menjalankan program aplikasi SVMRK pada sistem paralel MPI dibutuhkan sebuah file yang berisi beberapa data yang dibutuhkan untuk menjalankan metode iPIMRK yang diinginkan. Data-data tersebut misalnya: MMETHOD {metode MRK yang digunakan, misalnya: 22 artinya 2-*step* 2-*stage*, 33 artinya 3-*step* 3-*stage*}, VAR_METHOD {jenis koefisien yang digunakan misalnya, .TRUE. untuk koefisien berubah-ubah, .FALSE. untuk koefisien konstran}, TPRED {prediktor yang digunakan, misalnya: 1 untuk prediktor P1, 2 untuk prediktor P2, 3 untuk prediktor P3 dan 4 untuk prediktor P4}, ATOL {presisi digit toleransi absolut yang digunakan, misalnya 9 artinya 1.*d* - 09, khusus untuk tes problem *dense* dan *brusselator*: N {banyaknya variabel dari tes problem yang akan diselesaikan}, dan lain-lain. Keempat jenis aplikasi SVMRK, memiliki input file sendiri yaitu: `insvmrk0`, `incusp`, `inbruss`, dan `indensep0` masing-masing untuk DR_SVMRK, DR_CUSP, DR_SBRUSS dan DR_SDENSE.

Misalkan aplikasi SVMRK tersebut menggunakan MMTEHOD=33 sehingga bisa dijalankan pada tiga prosesor maka perintah untuk menjalankan masing-masing jenis aplikasi SVMRK adalah sebagai berikut:

- {untuk program DR_SVMRK} : `mpirun n0-2 ./vmrk.out <insvmrk0`
- {untuk program DR_CUSP} : `mpirun n0-2 ./cusp.out <incusp`
- {untuk program DR_SBRUSS} : `mpirun n0-2 ./bruss.out <inbruss`
- {untuk program DR_SDENSE} : `mpirun n0-2 ./dense.out <indensep0`

8. Hasil Eksperimen Numerik

Berikut ini adalah contoh hasil eksperimen numerik berupa *speed-up* dan presisi digit yang diperoleh dalam menyelesaikan beberapa tes problem *dense*, lihat Suhartanto[11] dengan variasi N=20, 30, 40, 50, 100, 150, 200, dan 250, menggunakan ATOL solusi eksak sebesar 1.d-11 dan ATOL solusi aproksimasi bervariasi dari 1.d-07, 1.d-08, 1.d-09 dan 1.d-10, MMETHOD=33, prediktor P3, VAR_METHOD=.TRUE. (*variabel coefficients*) dan .FALSE. (*fixed coefficients*) dengan hasilnya seperti terlihat pada tabel 3(a) dan 3(b). Ukuran evaluasi kinerja pada eksperimen ini menggunakan definisi dari Freeman[6].

Dari Tabel 3.(a) dapat dilihat bahwa hasil *speed-up* mulai terjadi pada beberapa kasus di N=30 dan secara signifikan terjadi untuk keseluruhan kasus mulai N=100. Untuk kasus-kasus N

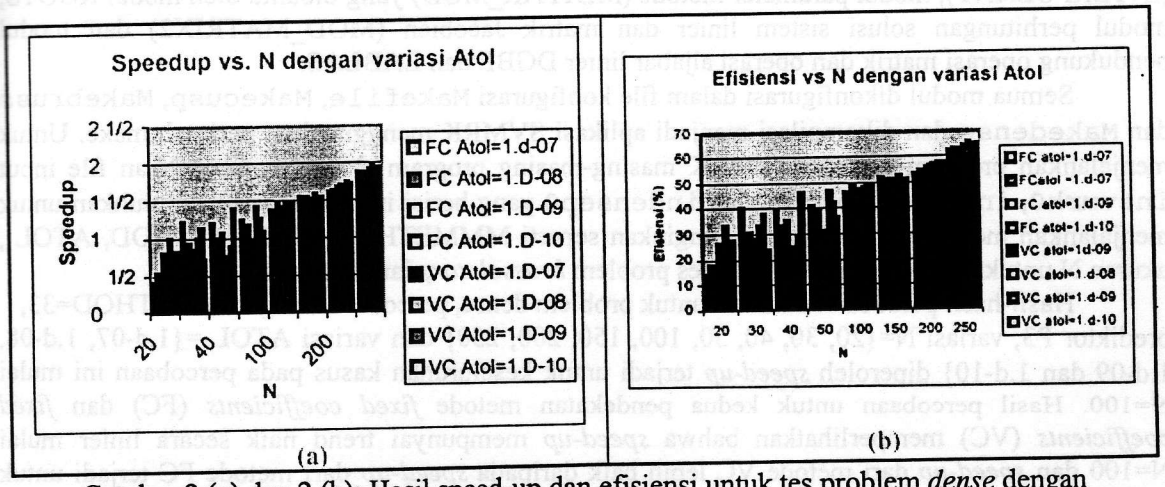
yang lebih kecil 200, secara umum perbandingan hasil *speed-up* antara metode dengan *variable coefficients*(VC) tidak lebih baik daripada metode dengan *fixed coefficients* (FC) karena nilainya berfluktuasi, tetapi mulai N=200 terlihat bahwa VC mempunyai *speed-up* yang lebih baik dari FC. Secara umum hasil *speed-up* memiliki trend naik secara linier khususnya mulai N=100, kesimpulan hasil yang sama secara otomatis juga berlaku untuk nilai efisiensi dan untuk lebih jelasnya dapat dilihat pada Gambar 2 dan Gambar 3.

Tabel 3.(a): Hasil *speed-up* untuk tes problem *dense* dengan MMETHOD=33, prediktor P3, variasi N, variasi ATOL, dan variasi VAR_METHOD

	Fixed Coefficients(FC)				*Variable Coefficients(VC)*			
	ATOL				ATOL			
N	**1,d-7**	**1,d-8**	**1,d-9**	**1,d-10**	**1,d-7**	**1,d-8**	**1,d-9**	**1,d-10**
20	0,42	0,54	0,35	0,80	0,24	1,00	0,64	0,82
30	1,22	0,79	0,93	0,85	0,75	1,00	1,14	0,72
40	1,19	1,00	0,81	1,06	0,71	0,75	0,89	1,39
50	1,23	1,11	1,18	0,89	0,95	1,41	1,26	1,10
100	1,38	1,37	1,47	1,42	1,42	1,45	1,49	1,46
150	1,55	1,53	1,54	1,53	1,54	1,59	1,50	1,54
200	1,60	1,64	1,64	1,68	1,70	1,71	1,74	1,73
250	1,88	1,89	1,87	1,88	1,92	1,96	1,95	1,95

Tabel 3.b: Hasil *presisi digit* untuk tes problem *dense* dengan MMETHOD=33, prediktor P3, variasi N, variasi ATOL, dan variasi VAR_METHOD

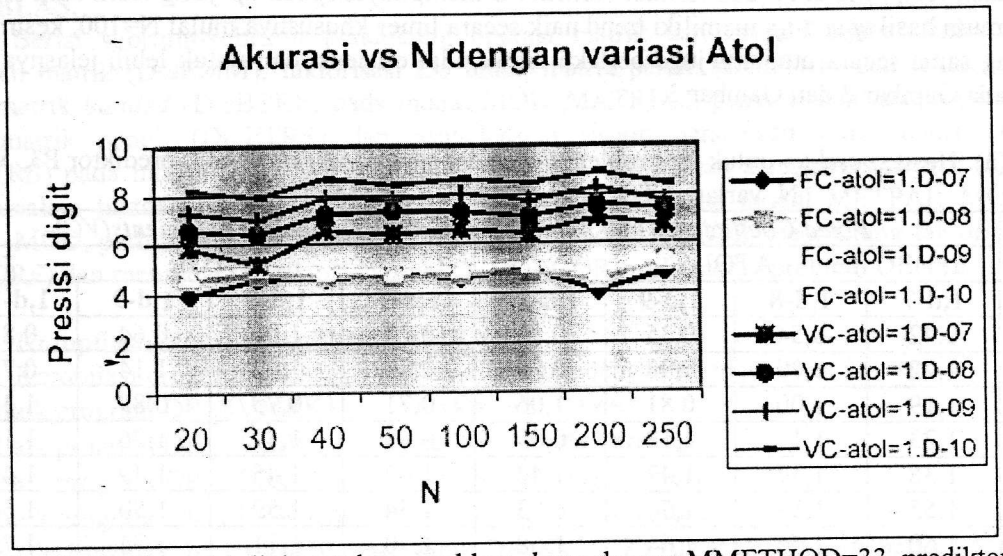
	Fixed Coefficients(FC)				*Variable Coefficients(VC)*			
	ATOL				ATOL			
N	**1,d-7**	**1,d-8**	**1,d-9**	**1,d-10**	**1,d-7**	**1,d-8**	**1,d-9**	**1,d-10**
20	3,95	4,59	4,78	5,24	5,90	6,59	7,28	8,21
30	4,60	4,80	5,07	5,37	5,21	6,42	7,01	7,92
40	4,63	4,70	4,97	5,30	6,58	7,24	7,89	8,71
50	4,77	4,69	4,89	5,12	6,41	7,34	7,81	8,43
100	4,64	4,78	5,02	5,32	6,60	7,31	8,04	8,65
150	4,99	4,84	5,19	5,48	6,43	7,10	7,75	8,47
200	4,07	4,50	4,84	5,21	7,03	7,62	8,25	8,90
250	4,86	5,27	5,47	5,65	6,74	7,40	7,72	8,44



Gambar 2.(a) dan 2.(b): Hasil speed-up dan efisiensi untuk tes problem *dense* dengan MMETHOD=33, prediktor P3, variasi N, variasi ATOL, dan variasi VAR_METHOD

Dari sisi presisi digit yang diperoleh berdasarkan Tabel 3.(b), terlihat bahwa metode VC secara signifikan selalu lebih baik dari pada metode FC untuk seluruh kasus, tetapi secara umum

nilai dari presisi digit tersebut berfluktuasi untuk keseluruhan kasus N dan variasi ATOL (tidak memiliki trend tertentu), untuk lebih jelasnya dapat dilihat pada Gambar 3.



Gambar 3: Hasil presisi-digit untuk tes problem *dense* dengan MMETHOD=33, prediktor P3, variasi N, variasi ATOL, dan variasi VAR_METHOD

9. Kesimpulan

Implementasi perangkat lunak SVMRK merupakan aplikasi dari metode iPIMRK yang dapat dijalankan pada sistem paralel MPI berbasis jaringan Linux. Program dibuat bersifat modular yang terdiri atas program implementasi paralelisasi berupa beberapa program *driver* sebagai modul pengontrol utama untuk beberapa kasus tes problem yang berbeda seperti: DR_SVMRK untuk berbagai persoalan *stiff* umum satu dimensi, DR_CUSP untuk persoalan CUSP berbentuk *banded*, DR_SBRUSS untuk persoalan Brusselator dua dimensi, dan DR_SDENSE untuk persoalan DENSE, dan tiga buah modul paralel (STAGES, FACTORS, dan SOLVES) sebagai pengontrol sekunder. Program integratornya berupa satu buah program *solver* SVMRK dibantu oleh sebuah integrator untuk menghitung langkah awal SIRK. Selain itu terdapat beberapa modul pendukung SVMRK lainnya seperti: Modul tes problem PROB_MOD, modul konfigurasi sistem matrik, modul ekstrapolasi matrik (MEXP_MOD), modul interpolasi ukuran langkah tetap (INTERPOLANT), modul parameter metode (METPAR_MOD) yang dibantu oleh modul ROOTS, modul perhitungan solusi sistem linier dan matrik Jacobian (MOD_MATRIX2) dan modul pendukung operasi matrik dan operasi aljabar linier DGBE dan LABLAS.

Semua modul dikonfigurasi dalam file konfigurasi `Makefile`, `Makecusp`, `Makebruss` dan `Makedense` dan dikompilasi menjadi aplikasi SVMRK menggunakan perintah make. Untuk menjalankan program aplikasi SVMRK masing-masing program *driver* membutuhkan file input `insvmrk0`, `incusp`, `inbruss` atau `indensep0` yang berisi informasi yang dibutuhkan untuk menjalankan metode iPIMRK yang diinginkan seperti MMMETHOD, VAR_METHOD, ATOL , ukuran N untuk tes problem *dense* dan tes problem *brusselator*, dan lain-lain.

Hasil-hasil percobaan numerik untuk problem dense, percobaan dengan MMETHOD=33, prediktor P3, variasi N={20, 30, 40, 50, 100, 150, 200, 250} dan variasi ATOL ={1.d-07, 1.d-08, 1.d-09 dan 1.d-10} diperoleh *speed-up* terjadi untuk keseluruhan kasus pada percobaan ini mulai N=100. Hasil percobaan untuk kedua pendekatan metode *fixed coefficients* (FC) dan *fixed coefficients* (VC) memperlihatkan bahwa *speed-up* mempunyai trend naik secara linier mulai N=100 dan *speed-up* dari metode VC lebih baik daripada *speed-up* dari metode FC terjadi untuk keseluruhan kasus pada percobaan ini mulai N=200. Dari sisi presisi digit ternyata metode VC memiliki hasil yang signifikan selalu lebih baik dari metode FC untuk keseluruhan kasus, tetapi hasil presisi digit ini selalu berfluktuasi untuk keseluruhan kasus, tidak memiliki trend tertentu.

10. Daftar Referensi

[1] Bustamam, A., *Implementasi metode iPIMRK untuk Menyelesaikan Persoalan Nilai Awal yang Stiff pada sistem paralel MPI-Linux*, Tesis S2 Magister Ilmu Komputer-UI, 2002

[2] Bustamam, A., Suhartanto, H. dan Basaruddin, T., *Implementasi metode iteratif paralel implisit multistep Runge-Kutta pada sistem paralel MPI-Linux,* Lokakarya Komputasi dalam Sains dan Teknologi Nuklir-XIII, BATAN, 2002

[3] Burrage,. K., dan Suhartanto, H. *Parallel iterated method based on multistep Runge-Kutta of Radau type for stiff problems*, Adv. Comput. Math., 1997, vol.7 , pp.59-77

[4] Burrage,. K., dan Suhartanto, H. *Parallel iterated method based on Variable-Step Multistep Runge-Kutta*, Submitted to Adv. Comput. Math., 1997

[5] Burley, J. C., *Using and porting GNU Fortran*, Free Software Foundation, Boston, U. S. A., 2001

[6] Freeman, L. dan Philips, C. , *Parallel Numerical Algorithms*, Prentice Hall International, UK, pp. 22-44, 1992

[7] Gropp, W., Lusk, E. dan Skjellum, *Using MPI: Portable Programming with Message Passing Interface*, MIT Press, 1995

[8] Koffman, E. B. dan Friedman, F. L., *Problem Solving and Structured Programming in Fortran 77*, Addison-Wesley Publishing Company, Inc., 1987

[9] Message Passing Interface Forum, *MPI: A message passing interface standar*, Int. Journal of Supercomputer, 1994, vol 8(3/4), pp. 159-416, special issue on MPI

[10] Nyhoff, L. dan Leestma, S., *Fortran 90 for Engineers and Scientist*, Prentice Hall, NJ, 1997

[11] Suhartanto, H. dan Burrage, K., *S-PMRK: Parallel Stiff ODE solver based on Iterated Variable-Step Multistep Runge-Kutta*, Submitted to Adv. Comput. Math.,1997

[12] Suhartanto, H., *Parallel Iterated Techniques Base on Multistep Runge-Kutta Method of Radau Type*, Ph.D Thesis, University of Queensland, Brisbane, Australia, 1997